МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Код Хаффмана

Вариант 6

Студент гр. 8304 ___ Птухов Д. А.

Преподаватель ___ Фиалковский М. С.

Санкт-Петербург

2019

Цель работы.

Получить опыт работы с кодом Хаффмана.

Постановка задачи.

Декодирование кода Хаффмана.

Описание алгоритма.

1) Считывание осуществляется при помощи конструкции:

```
while (std::getline(in, currentFileString))
```

2) После считывания всех строк каждая из них проверяется на корректность, в случае ошибки выводится соответствующее сообщение.
3) После успешной проверки входных данных при помощи пробега по строке и поиска очередного рассматриваемого кода в ранее сформированном дереве осуществляется декодирование строки.

Спецификация программы.

Программа предназначена для декодирования кода Хаффмана.
Программа написана на языке C++.

Описание функций и структур данных.

1) Для формирования словаря, имеющего форму символ : двоичный код символа, была реализована функция dictCreation, которая при помощи регулярных находит очередную строку вида (символ код) и записывает найденные данные в словарь попутно осуществляя проверку появления коллизий.

2) Для формирования дерева содержащегося в себе коды символов была реализована функция formHaffmanTree, которая при помощи пробега по очередному коду из ранее сформированного словаря создает дерево Хаффмана.

3) Для декодирования ранее считанных данных была написана функция decode, которая пробегая по строке формирует очередной код и проверяет его наличие в дереве. В случае его наличия записывает соответствующий коду символ в результирующую строку.

Тестирование.

Таблица 1 – Результаты тестирования программы

Input	**Output**
00101110001 ((H 001)(E 01)(L 1)(O 0001))	HELLO
001 ((A 001))	A
10000 ((A 1000)(B 0))	AB
NIUGEBIGQNBOINBLCMANlknalJBJb ((C 1000))	Code contains incorrect symbols
0000 ((A 0)(A 1))	Collision
0101111000 ((A 100)(B 200))	Your map-string is incorrect
000011 ((U 1)(B 01)	Your map-string is incorrect

Выводы.

В ходе работы был получен опыт работы с кодом Хаффмана.

Приложение А. Исходный код программы.

Source.cpp

```
#include "Header.h"

//здесь мог быть нерелевантный комментарий
bool checkMapStringFormCorrection(std::string const& checkString)
{
	std::regex pattern("\\(\\(\\w [01]+\\))+\\)");
	return std::regex_match(checkString, pattern);
}

bool checkCodeCorrection(std::string const& checkString)
{
	return std::find_if(checkString.begin(), checkString.end(), [](char c) {return c != '0' &&
c != '1'; }) == checkString.end();
}

ReturnCode checkDataCorrection(std::string const& code, std::string const& charactersCodes)
{
	bool resultCodeCorrectionCheck = checkCodeCorrection(code);
	if (!resultCodeCorrectionCheck)
		return ReturnCode::IncorrectSymbols;

	bool mapStringFormCorrection = checkMapStringFormCorrection(charactersCodes);
	if (!mapStringFormCorrection)
		return ReturnCode::IncorrectMapStringForm;

	return ReturnCode::Correct;
}

StringVector readFileData(std::ifstream& in)
{
	StringVector fileData;
	std::string currentFileString;
	while (std::getline(in, currentFileString))
	{
		if (currentFileString.back() == '\r')
			currentFileString.erase(currentFileString.end() - 1);
		fileData.push_back(currentFileString);
	}

	return fileData;
}

ReturnCode formHaffmanTree(std::string const& code, char character, std::shared_ptr<Node>& head)
{
	std::shared_ptr<Node> headCopy(head);

	for (char element : code)
	{
		if (headCopy->character != 0)
			return ReturnCode::BadPreffixForm;

		if (element == '0')
		{
			if (headCopy->left == nullptr)
			{
				auto newElement = std::make_shared<Node>();
				headCopy->left = newElement;
			}

			headCopy = headCopy->left;
		}
```

```
			else
			{
				if (headCopy->right == nullptr)
				{
					auto newElement = std::make_shared<Node>();
					headCopy->right = newElement;
				}
				headCopy = headCopy->right;
			}
		}
		if (headCopy->character != 0)
			return ReturnCode::BadPreffixForm;

		headCopy->character = character;

		return ReturnCode::Correct;
}

CharactersCodeMap dictCreation(std::string& parsedString, ReturnCode& creationResult)
{
		std::regex pattern("\\(\\w [01]+\\)");
		std::smatch match;
		std::map<char, std::string> dict;

		while (std::regex_search(parsedString, match, pattern) != 0)
		{
			std::string matchedBrackets(match.str());

			std::string characterCode(matchedBrackets.find(" ") + 1 + matchedBrackets.begin(),
matchedBrackets.find(")") + matchedBrackets.begin());
			char character = matchedBrackets[1];
			if (dict.find(character) != dict.end())
			{
				creationResult = ReturnCode::CollisionError;
				return dict;
			}

			dict[character] = characterCode;

			parsedString.erase(match.position() + parsedString.begin(), match.length() +
match.position() + parsedString.begin());
		}

		return dict;
}

//Возвращаемое функции зависит от того стоит ли продолжать поиск или нет
bool findCodeInTree(std::shared_ptr<Node>& head, std::string const& code, char& character)
{

		if (code.back() == '1')
		{
			if (head->right == nullptr)
				return false;

			else
			{
				if (head->right->character != 0)
				{
					character = head->right->character;
					return true;
				}
				else
				{
					head = head->right;
					return true;
				}
```

```
			}
		}
		else
		{
			if (head->left == nullptr)
				return false;
			else
			{
				if (head->left->character != 0)
				{
					character = head->left->character;
					return true;
				}
				else
				{
					head = head->left;
					return true;
				}
			}

		}
}

DecodePair decode(std::string const& code, std::shared_ptr<Node> const& head)
{
	std::string currentCheckedCode;
	std::string decodeResult;

	std::shared_ptr<Node> startPoint(head);

	for (char element : code)
	{
		currentCheckedCode += element;

		char character = 0;
		bool findResult = findCodeInTree(startPoint, currentCheckedCode, character);
		if (findResult == false)
			return { "", false };

		if (character != 0)
		{
			decodeResult += character;
			startPoint = head;
			currentCheckedCode.clear();
		}
	}

	return { decodeResult, true };
}

int main(int argc, char** argv)
{
	if (argc > 2)
	{
		std::ifstream in(argv[1]);
		if (!in.is_open())
		{
			std::cout << "Incorrect input file\n";
			return 0;
		}

		std::ofstream out(argv[2]);
		if (!out.is_open())
		{
			std::cout << "Incorrect output file\n";
			return 0;
		}
```

```
		auto fileData = readFileData(in);

		if (fileData.size() < 2 || fileData.size() % 2 != 0)
		{
			std::cout << "File should consist of an even number of lines (0 is not even)\n";
			return 0;
		}

		for (auto it = fileData.begin(); it != fileData.end(); it += 2)
		{
			std::string checkCode(*it);
			std::string checkMapStringForm(*std::next(it));

			ReturnCode checkResult = checkDataCorrection(checkCode, checkMapStringForm);
			if (checkResult == ReturnCode::IncorrectSymbols)
			{
				out << "Code contains incorrect symbols\n";
				continue;
			}
			if (checkResult == ReturnCode::IncorrectMapStringForm)
			{
				out << "Your map-string is incorrect\n";
				continue;
			}

			ReturnCode creationResult = ReturnCode::Correct;
			auto dict = dictCreation(checkMapStringForm, creationResult);
			if (creationResult == ReturnCode::CollisionError)
			{
				out << "Collision\n";
				continue;
			}

			auto head = std::make_shared<Node>();
			bool isGoodPrefixForm = true;

			for (auto i : dict)
			{
				//second - код символа, first - сам символ
				//Я считаю это нужным комментарием т.к. из-за непонятных имен полей first и second нам было запрещено пользоваться std::pair
				ReturnCode formResult = formHaffmanTree(i.second, i.first, head);
				if (formResult == ReturnCode::BadPreffixForm)
				{
					isGoodPrefixForm = false;
					break;
				}
			}

			if (isGoodPrefixForm == false)
			{
				out << "Bad prefix form\n";
				continue;
			}

			DecodePair result = decode(checkCode, head);
			if (result.decodeResult == false)
			{
				out << "Your code is incorrect\n";
				continue;
			}

			out << result.value << "\n";
		}
	}
```

```
        return 0;
}
```

Header.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <algorithm>
#include <variant>
#include <memory>
#include <map>
#include <regex>

using StringVector = std::vector<std::string>;
using CharactersCodeMap = std::map<char, std::string>;

enum class ReturnCode
{
	IncorrectSymbols,
	IncorrectMapStringForm,
	BadPreffixForm,
	CollisionError,
	Correct
};

struct Node
{
	Node() = default;

	char character = 0;

	std::shared_ptr<Node> left;
	std::shared_ptr<Node> right;
};

struct DecodePair
{
	DecodePair() = default;

	std::string value;
	bool decodeResult = true;
};

void dynamicFileReading(std::ifstream&);
```